Automatic Generation of Interview Questions for Human Experts to Acquire Database Keys with Perfect Precision and Recall

Abstract

Database constraints are of utmost importance for the correct modeling, high quality, and efficient operation of databases. Keys, in particular, are fundamental to entity and referential integrity which provide sound and efficient access to data. Acquiring the set of database constraints that express rules of the underlying application domain is therefore essential. Without computational support to involve teams of human experts in the acquisition process, it is very likely that some database constraints may be perceived incorrectly as rules, decreasing precision, or some rules may not be recognized as such, decreasing recall. We propose a methodology for providing computational support towards the acquisition of rules with perfect precision and recall, which combines and extends previous work in this area. We exemplify the methodology on the most important class of database constraints, keys, for which we develop and analyze a suite of algorithms that generates Boolean questions asked to teams of human experts. We show that enumerating all questions that can ensure perfect precision and recall of the acquisition process is as hard as the famous transversal hypergraph problem. Experiments with synthetic and real world examples illustrate how well the algorithms in our suite address different synthetic and real-world distributions of keys, and how integrating previous approaches such as key mining and common sense can make the search space feasible for traversal and investigation.

CCS Concepts

• **Human-centered computing** → *Interaction design*; • **Information systems** → **Relational database model**; **Integrity checking**; **Inconsistent data**; **Database utilities and tools**.

Keywords

Algorithm, Database key, Experiment, Requirements acquisition

ACM Reference Format:

. 2018. Automatic Generation of Interview Questions for Human Experts to Acquire Database Keys with Perfect Precision and Recall. In . ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Database constraints form a declarative approach towards ensuring that the semantics of application domains is represented adequately within database systems [11, 16, 46]. In fact, constraints restrict database instances to those that satisfy all the rules of the underlying application domain. As a simple example, just as no one in the real world should have multiple valid passport numbers of the same country, this rule should be declared, monitored, and enforced within database systems that model this information. Without sound representations, the two most essential database tasks are compromised: update operations may introduce data that violate data integrity, and any analytical method applied to invalid data will not produce insight. Hence, identifying those database constraints that need to be declared and enforced on a database schema is of utmost importance [11, 16, 46]. We refer to those constraints as *rules* or *meaningful constraints*, in contrast to those constraints that can be violated by some real-world scenario and, therefore, do not express rules that need to be declared.

While database research has surfaced over one hundred different classes of database constraints [12, 50], the perhaps most important classes are domain constraints, keys and foreign keys. These are the only classes that enjoy native support in database systems to ensure domain integrity, entity integrity, and referential integrity, respectively [11, 12]. This article will focus on keys, which is a finite set K of columns such that no two different records in a table can have values that match in all columns of K [35]. Due to their role in implementing entity integrity, keys are essential for all data models, including relational [35], conceptual [51], object-relational [30], XML [9], RDF [33], temporal [55], spatial [23], probabilistic [8], and graph [2] models. They are fundamental in many classical areas of data management, including data modeling, database design, indexing, and query optimization. Knowledge about keys enables us to (i) uniquely reference entities across data repositories, (ii) minimize data redundancy at schema design time to process updates efficiently at run time, (iii) provide better selectivity estimates in cost-based query optimization, (iv) provide a query optimizer with new access paths that can lead to substantial speedups in query processing, (v) allow the database administrator to improve the efficiency of data access via physical design techniques such as data partitioning or the creation of indexes and materialized views, and (vi) provide new insights into application data. Modern applications raise the importance of keys further. They can facilitate data integration [10], help with the detection of duplicates and anomalies [40], provide guidance in repairing and cleaning data [25], enhancing the reliable use of large language models and AI agents for data processing tasks [34]. The discovery of keys from data is one of the core activities in data profiling [1].

We address the following fundamental research question:

How can we acquire the set of meaningful keys on a given schema?

Striking real-world examples occur whenever surrogate identifiers are used without analyzing which other keys are meaningful. In practice, the identifiers often replace keys altogether. However, not only can surrogate ids alone not address the research question but they affect data management negatively. In fact, without a meaningful key, there is no guarantee that multiple ids are attributed to the same entity. The latter scenario causes duplicates,

Conference'17, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

which has led to its own research area: entity resolution or data deduplication [40]. An impactful showcase is the attribution of multiple health identifiers to the same individual, meaning that health histories become only partially accessible to physicians, which may lead to poor decisions with dire consequence. Similar examples include products and customers with multiple ids, causing wrong recommendations by data-driven analysis.

The research question has remained challenging since the beginning of database research. Even when it is known what collection of columns form the underlying database schema, the set of meaningful keys may only be known partially. In fact, some database schemata may not have a key, requirements may evolve, or may not have been researched well enough. The set of meaningful keys on a given schema ought to enjoy perfect precision and recall. Perfect precision means we do not specify any key that is meaningless. If we did specify some meaningless key, then this would prevent some valid data from entering the database. Perfect recall means we specify every key that is meaningful. Failure to do so would allow some invalid data to enter the database. Identifying the set of meaningful keys is therefore time-consuming and never ending. In fact, for a table with n columns, there may be up to $\binom{n}{\lfloor n/2 \rfloor}$ minimal keys, so even if we guessed perfectly what the set of meaningful keys are, there may still be exponentially many. The search space that needs traversal consists of 2^n column sets. We may restrict this space to sets of size up to some arity, but this will compromise perfect recall in case there is some meaningful key with some higher arity. A lot of effort has gone into the design of algorithms that mine constraints of a given class from a given data set, including keys, functional dependencies, foreign keys and inclusion dependencies, and many other classes [1]. However, the output of these algorithms is simply a set of constraints that holds on the input data set, and it is not clear which of these constraints constitute meaningful rules. While approximate mining improves recall, it also decreases precision, meaning that more constraints of the mining output need to be validated by human experts. Nevertheless, apart from the given schema, the input to our research question may consist of meaningful keys that have been validated before, such as the keys specified on the given schema, or some keys that have been validated by human experts from the output of data profiling.

Example 1.1. As preview to our framework consider the largest table of the TPC-H benchmark but with a reduced schema of only four columns: L(inenumber), O(rderkey), P(artkey), and S(hipdate). The primary key of the table is {L, O}, but {O, P} constitutes another meaningful minimal key. Fig. 1 shows the underlying Boolean lattice of column sets that needs traversal to identify these meaningful minimal keys, with the latter shown in dark blue. Super-keys are marked in light blue, while anti-keys (column sets that are not keys) are marked in magenta and maximal anti-keys in red. The numbers next to the column sets indicate the order in which they are traversed, which in this case follows a top-down, breadth-first strategy. Here, the primary key is already input as a predetermined minimal key to the interview process. Hence, none of its super-keys need traversing. The bottom figure shows the result of the process with the set of meaningful minimal keys fully identified, together with a database instance that satisfies all keys and violates all anti-keys, which is also known as an Armstrong database.

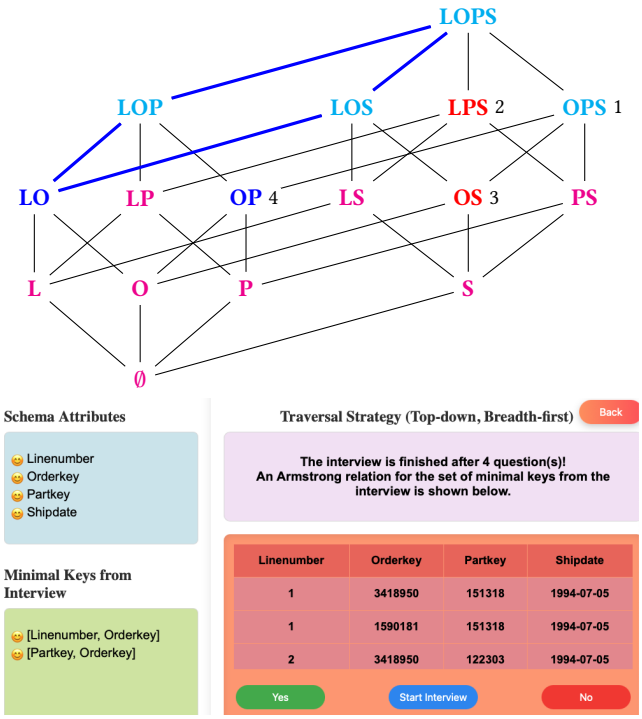


Figure 1: Top-down breadth-first traversal with primary key as input for interviewing the set of minimal keys, and final result of tool shown with an Armstrong database

As our main contribution, we propose a new methodology for addressing the fundamental research question above. It complements and integrates previous work on data profiling, data sampling, and constraint acquisition. The opportunities to improve precision and recall of the acquisition process are illustrated in Fig. 2. Here, set R denotes the target set of meaningful keys while C denotes the set of keys currently perceived as meaningful by the team of human experts. Our algorithms interactively generate Boolean questions to interview the experts. If they answer *No*, the key in question is meaningful, while a positive answer means it is not. Depending on how the set of meaningful keys is distributed across the lattice of column sets, different traversal strategies generate interview questions in different order. The point of the process is to traverse all candidates effectively and as efficiently as possible given the exponential search space and potentially exponential number of meaningful keys. While common sense goes a long way in this process, algorithms with complementary traversal strategies can effectively explore different parts of the search space whenever time and resources are limited. Our **contributions** are as follows.

(1) Conceptually, we propose a new methodology for acquiring the set of meaningful keys on a given schema. The methodology brings together extensive previous work on data profiling, Armstrong databases, and constraint acquisition, but adds a new dimension that focuses on the systematic traversal and validation of potential keys by human experts. While we introduce the methodology for the class of keys, it can be applied to any class of database

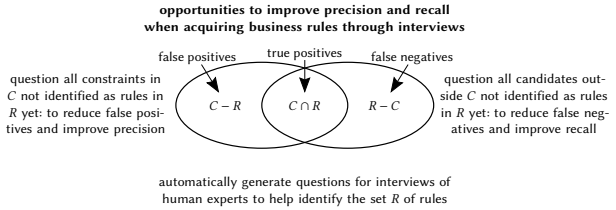


Figure 2: Opportunities to improve the precision and recall when acquiring meaningful database keys

constraints, and we propose different dimensions for future research. (2) Computationally, we propose a suite of four algorithms that generates interview questions for human experts in different order, each aligned with different distributions of the keys they target. These include combinations of bottom-up, top-down, breadth-first, and depth-first approaches that traverse the underlying lattice of column sets. In general, we show that the enumeration of the interview questions is computationally as hard as enumerating the edges of a hypergraph transversal. Hence, it will be an open question whether an algorithm exists for these two problems that is polynomial in the output. (3) Experimentally, we have made our algorithm suite publicly available as a tool for directly acquiring database keys. We applied the implementations to conduct several experiments. We show which distributions of minimal keys are addressed by which algorithms, which is demonstrated on synthetically created sets of minimal keys, sets of minimal keys that hold on real world data sets, and real world sets of minimal keys. Within the bounds of an intractable computational problem, the experiments provide insight how well the algorithms scale in the number of interview question asked. Adding predetermined minimal keys, and removing columns from the schema that are unlikely to occur in any key are demonstrated as practical techniques to make the acquisition process feasible.

Empirical research shows that over half the errors in system development are due to requirements issues [17, 32, 39], and these errors are the most frequent reason for project failures [17, 47, 48]. Furthermore, the cost of fixing errors rises dramatically throughout development; correcting a defect after implementation can be over 100 times more expensive than fixing it during requirements analysis [6]. Therefore, focusing quality assurance on the requirements analysis phase is most effective for catching and preventing errors [56]. Apart from raising the quality, efficiency and effectiveness of database systems by acquiring meaningful keys (and other constraints) with perfect precision and recall, we can therefore also see the financial benefits resulting from our methodology. Researching how computational resources for constraint acquisition can be minimized will drive these benefits further.

Organization. Section 2 positions our contributions within related work. Our methodology is detailed in Section 3. Complexity results establish the computational hardness of the underlying decision problems in Section 4 to provide context for our contributions. Our suite of algorithms is described in Section 5. Experimental results are presented in Section 6, before drawing conclusions and outlining future work in Section 7.

2 The Missing Piece in Constraint Acquisition

We summarize previous work on the acquisition of database constraints, in particular keys. These include the areas of data profiling and Armstrong databases, and how they work together.

2.1 Discovery of Constraints from Data

This area is also known as *dependency inference*, *dependency discovery* or *data profiling* [1]. The input is a database instance and a class C of database constraints, with the goal to compute the set of database constraints in C that hold on the database instance. Different classes of constraints and types of database instances have been considered, such as the class of functional dependencies over database relations with missing data. For the most popular classes of constraints, the computational complexity of the associated decision problem has been established [4]. For example, deciding whether there is some key of size at most s is both NP-complete [35] and $W[2]$-complete in s [4]. Furthermore, the search space is exponential in the number of columns, and the solution space is worst-case exponential as well [13]. The general tractability of discovering keys (and other constraints) from data has therefore strong computational limits. However, a broad range of deep algorithms [5, 20, 28, 37, 52, 54] and tools [42, 53] have been proposed, allowing to compute solutions to the discovery problem for large instances, even with many rows and columns.

Different search strategies and kinds of algorithms have been devised: column-efficient [21], row-efficient [27], hybrid [43, 54], approximate [44], incremental [7], distributed [45], all with the primary and often only focus on efficiency. An area that has not received much attention yet is how the output of discovery algorithms can be explored or utilized, apart from some ranking techniques and holistic prioritization [28, 52]. Discovery algorithms do typically not distinguish between constraints that hold accidentally and those that are meaningful. In other words, the precision and recall of automated rule discovery from data is not a primary concern of discovery algorithms. Research in this area is in its infancy due to the challenges it presents: data is typically not of good quality exhibiting inaccuracy, incompleteness, inconsistency, untimeliness. This affects precision and recall, but even with data of perfect quality we still need human experts to identify the few rules among the many constraints that have been discovered. Even the best mining tools cannot provide decisions, which must be made by human experts. Hence, we need to involve humans-in-the-loop.

2.2 Creating Armstrong Databases

Perfect example databases have been shown to improve the recall of identifying meaningful database constraints [31, 36, 38]. Given a finite set C of constraints from a class $\mathcal{C}$, a database instance db is said to be *Armstrong (for C with respect to $\mathcal{C}$)* if, for every constraint $c \in C$, db satisfies c if and only if c is implied by C [3, 19]. Here, implication refers to the fact that every database instance that satisfies all the constraints in C also satisfies c. It is a perfect, user-friendly representation of C since it satisfies all constraints in C but violates all constraints not implied by C [36]. Hence, if C denotes the set of constraints that is currently perceived as the rules in $\mathcal{C}$, then an Armstrong database can be used to validate the meaningfulness of every constraint c in $\mathcal{C}$: If the Armstrong database satisfies c,

then c is also perceived as meaningful; and otherwise c is not perceived as meaningful. Constraints that are incorrectly perceived as meaningless are violated by any Armstrong database. Human experts may spot these violations and can point them out to data professionals, who can specify the constraints as rules accordingly. Human experts may also edit the Armstrong database to make it an instance that the experts perceive as perfect representative for the rules in C. Discovery algorithms may be used subsequently to mine the set R of rules from that instance. Research in this area has only seen first steps, but they are promising. Nevertheless, there are challenges and limitations. Armstrong databases need to violate all the constraints not implied by the given set of constraints and, therefore, contain as many records as necessary to exemplify all of these violations. As a simple example, the minimum size of an Armstrong relation for an arbitrary system of minimal keys over n columns has lower bound of $\frac{1}{n^2}\binom{n}{\lfloor n/2 \rfloor}$ [14]. It is notoriously difficult to ensure minimality by general algorithms that compute Armstrong databases, with general upper bounds being quadratic in the minimal size [3, 36]. Hence, the sheer number of records makes it challenging for human experts to spot inconsistencies, and even more difficult to correct them manually. In general, only few classes of integrity constraints actually enjoy Armstrong databases, that is, there is no guarantee that for members of a given class of constraints an Armstrong database exists [18]. While the class of keys does enjoy Armstrong databases [18], the biggest limitation is the inability to direct the focus of human experts to particular constraints in question.

2.3 Constraint Acquisition Prompted by Experts

The limitations of Armstrong databases have sparked other work on constraint acquisition [26] suggesting the generation of small samples that satisfy the given set C of constraints but only violate the given constraint c in question. In that approach it is left to human experts to determine which constraints will be questioned. In contrast, we propose algorithms that implement different traversal strategies ensuring all candidate constraints have been exhausted.

2.4 Summary

Different approaches have been considered to facilitate the acquisition of meaningful database constraints. Missing is computational support to efficiently and effectively traverse the search space of candidate constraints, and the integration of previous approaches in one methodology that enables human experts to identify meaningful constraints with perfect precision and recall.

3 Methodology and Lines of Research

This section will bring together previous directions of research in a methodology to acquire meaningful database constraints of a given class with perfect precision and recall. While the methodology itself is not limited to a specific class, we will focus on database keys in subsequent sections to provide a proof of concept for the methodology using the most important class of database constraints. We will also outline different dimensions for future research. We start with some technical definitions to fix terminology and notation.

3.1 Preliminaries

Relational database systems use relations as the prime mechanism to model an application domain. As any data model, it distinguishes between schema and instance level.

A *table schema* is a finite, non-empty set T of *attributes*, representing columns of a table. Each attribute A has a domain *dom(A)*, which represents the domain values that can appear in column A. A *database schema* is a finite, non-empty set $\mathcal{S}$ of table schemata.

A *record* over table schema T is a function r that assigns to each attribute A of T a value $r(A)$ from the domain $dom(A)$. The *projection* $t(X)$ of r onto a set $X \subseteq T$ is the function r restricted to columns in X. A *table* over table schema T is a finite set t of records over T. A *database instance* i over database schema $\mathcal{S}$ assigns to every table schema $T \in \mathcal{S}$, a table $i(T)$ over T. We use $\mathcal{I}(\mathcal{S})$ to denote the set of all database instances i over database schema $\mathcal{S}$.

A *database constraint* c over database schema $\mathcal{S}$ is a function $c\colon \mathcal{I}(\mathcal{S}) \to \{0,1\}$ that maps every database instance $i \in \mathcal{I}(\mathcal{S})$ to either 0 or 1. Intuitively, $c(i) = 1$ represents the case where c is satisfied by i, while $c(i) = 0$ represents the case where c is violated by i. For a given set C of database constraints, we say that that a database instance i *satisfies* C if and only if i satisfies every constraint $c \in C$. The set C of database constraints over schema $\mathcal{S}$ classifies instances in $\mathcal{I}(\mathcal{S})$ into the set $C(\mathcal{S}) = \{i \in \mathcal{I}(\mathcal{S}) \mid \forall c \in C(i \text{ satisfies } c)\}$ of instances that satisfy C, and those that do not.

Let $\mathcal{M} \subseteq \mathcal{I}(\mathcal{S})$ denote the set of database instances that represent real world scenarios of the application domain. Then the *problem of constraint acquisition* is to identify the set C of constraints over $\mathcal{S}$ such that $C(\mathcal{S}) = \mathcal{M}$. Typically, the problem restricts our attention to constraints from a given class $\mathcal{C}$.

For a set $C \cup \{c\}$ of constraints in class $\mathcal{C}$ over schema $\mathcal{S}$, C *$\mathcal{C}$-implies* c if and only if every database instance $i \in \mathcal{I}(\mathcal{S})$ that satisfies C also satisfies c. Intuitively, by specifying the constraints in C explicitly, we also specify any constraint implied by C implicitly.

For a given set C of database constraints in class $\mathcal{C}$ over schema $\mathcal{S}$, a database instance db over $\mathcal{I}(\mathcal{S})$ is said to be C-Armstrong if and only if it satisfies C and violates every constraint c in $\mathcal{C}$ over $\mathcal{S}$ that is not C-implied by C.

In this work we focus on the class $\mathcal{C}$ of keys over table schemata $T \in \mathcal{S}$. A *key* over T is a set $K \subseteq T$. A table t over T *satisfies the key* K if and only if there are no two different records $r, r' \in t$ such that r and r' have values matching on all the columns in K, that is, there is some $A \in K$ such that $r(A) \neq r'(A)$. For a set $C \cup \{K\}$ of keys over T, C implies K if and only if there is some set $K' \in C$ such that $K' \subseteq K$. Hence, key-implication simply requires checking a subset relationship. This motivates the following terminology. A key K over T is said to be *minimal* for a set C of keys over T if and only if there is no key K' that is a proper subset of K, that is, $K' \subset K$. While we distinguish between minimal keys and keys, other work uses the word key to represent minimal keys and the word super-key to refer to keys that are minimal or not. We may also use the word *candidate key* to refer to a minimal key. The class of keys enjoys Armstrong databases.

3.2 General Methodology

We start with the problem statement using Fig. 2. Given a class $\mathcal{C}$ of database constraints, a database schema $\mathcal{S}$, and a finite set C of

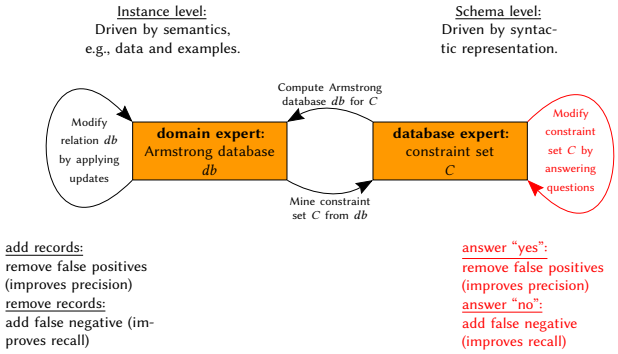


Figure 3: Methodology to Acquire the Set of Meaningful Database Keys with Perfect Precision and Recall

database constraints in class C, the goal is to provide computational support for a team of human experts in revising C such that the final revision coincides with the set R of rules in class C.

The team of human experts may comprise stakeholders for the target database, such as users, domain and database experts. The set C of database constraints is an evolving set of those constraints in C that the team currently perceives as meaningful. It may consist of constraints identified as meaningful by previous stages of the process, or part of the output of other approaches, including the result of discovery algorithms, the inspection of Armstrong databases, or constraint acquisition through manual prompting, see Section 2. C may be empty at the beginning of the process.

Fig. 3 illustrates the overall methodology we propose for acquiring the set R of meaningful database constraints with perfect precision and recall. The novel part is highlighted in red, while the remaining parts bring together the approaches reviewed in Section 2. The modification of the constraint set C is driven by the automatic generation of some Boolean question whether it is possible for the constraint under investigation to be violated, which is exemplified by a small database instance i. The Boolean question is represented in natural language and the instance i consists of records made up by synthetic or real world values when they exist. While this borrows the idea of constraint acquisition prompted by humans [26], the novelty is the automatic and interactive generation of these questions through an exhaustive traversal of the remaining search space. If the team of experts answers *No*, then the constraint in question will be added to C, otherwise C will remain unchanged. Depending on the answer and an underlying traversal strategy, the generation of the next question is triggered. This continues until all constraints have been considered, or the human experts decide to use other techniques. Section 5 will propose different traversal strategies, each valuable for particular distributions of minimal keys.

3.3 Methodology Applied to Keys

In the specific case of keys K in question, the sample instance i can always be a table over the given table schema T, where i consists of two records in which for every column of T, the two records have matching values if and only if the column is in K. In essence, we are asking the expert team whether it is possible that a real-world table exists which violates K. If not, K should be added to the current set C of database constraints perceived meaningful. Note that the traversal strategies are always such that no key will ever be in question if it is implied by C, or has been invalidated either explicitly or implicitly through previous questions. Indeed, K is implied by key set C if and only if there is some key $K' \in C$ such that $K' \subseteq K$. This means, once a key K is added to C we do not need to question any of its super-keys and prune the search space accordingly. In addition, once K has been confirmed that it is not a rule, none of its sub-keys $K' \subseteq K$ needs to be questioned (if any of them were meaningful, K would be implied and a rule, too), and the search space can be pruned accordingly.

3.4 How Precision and Recall are Improved

Apart from introducing computational support to traverse the search space of interview questions, the methodology also utilizes all previous techniques discussed in Sec 2. We will describe now how the methodology improves the precision and recall towards the goal of identifying the set of meaningful constraints.

Firstly, answering "No" means that the constraint c in question should be added to C. Since c was previously not included in C but will be included as a result of the interview, we have removed a false negative, which improves the recall of C.

Secondly, the human experts may prompt a particular question out of the order suggested by the traversal strategy, which integrates prompted constraint acquisition [26]. This may include a constraint c in C itself. This happens when the expert team finds it questionable whether c is meaningful. If the answer to questioning c is "Yes", then c will be discarded from C, therefore removing a false positive, which improves the precision of C.

These two interview techniques take part on the schema level, are driven by the syntactic representation of the constraints in the given class, and the investigation is led by database experts. However, the domain and user experts play an invaluable part during the interviews since each question presents a sample database that is judged by them under guidance by the database experts. The two techniques adopt a local point of view which directs the focus of the expert team on a specific constraint in question.

Fig. 3 shows how the methodology promotes the switch to an instance-level approach, driven by the semantics of the constraints and illustrated on actual data. That approach is global because the constraint set C is represented by an Armstrong database db. Users and domain experts lead the approach by modifying the Armstrong database. However, database experts play an invaluable role since they point out how modifications impact the constraint set.

Thirdly, the expert team may add a record to the sample database that may invalidate a constraint c that was previously valid, that is, $c \in C$. As a consequence, c would be discarded from C, effectively removing a false positive, which improves precision.

Finally, the expert team may delete a record from the instance, which may validate a constraint c that was previously invalid, that is, c was previously implied by C. Hence, c will need to be included in C, effectively adding a false negative, which improves recall.

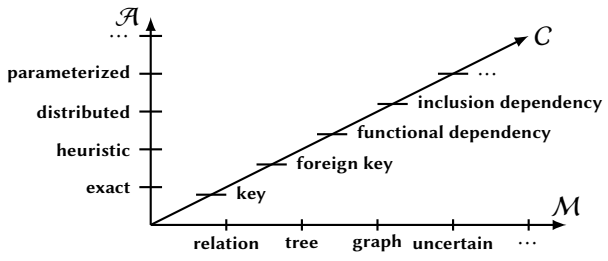


Figure 4: Dimensions of Research in Constraint Acquisition

As illustrated in Fig. 3, the methodology can switch back to the schema-level approach, by mining the set C of constraints represented by the modified Armstrong database *db*.
This process may be repeated until the team is satisfied, resources have been exhausted, or a final decision is due. The process is deliberately flexible to accommodate different inputs and limitations in terms of resources available. For instance, the process can be terminated early, the search space can be restricted to constraints of a particular size, or real world records may be sourced as part of the interview or Armstrong databases.

3.5 Dimensions of Research

The methodology gives scope to a rich landscape of research directions. One may select any combination of dimensions from algorithmic approaches $\mathcal{A}$, the class $\mathcal{C}$ of constraints, and the model $\mathcal{M}$ of data. Each combination defines a specific research setting in this area, as illustrated in Fig. 4.

Algorithmic approaches may consider traversal strategies such as bottom-up, top-down, depth-first, breadth-first, or mixed ones; but also exact, heuristic, approximate, randomized, distributed and parameterized algorithms. Classes of constraints may include keys, foreign keys, functional dependencies, inclusion dependencies, join dependencies, order dependencies, etc. [51]. Models may include relational, bag, tree, graph, spatial, temporal and uncertain data.

Here we start with exact traversal strategies for the class of keys over database relations.

4 Computational Complexity

This section will explore the parameterized and enumeration complexity of problems associated with the acquisition of meaningful database keys. We assume basic familiarity with complexity theory [41], and will provide a brief introduction to parameterized complexity theory [15].

A *parameterized problem (PP)* Π is a subset of $\Sigma^* \times \mathbb{N}$, for an alphabet Σ. For tuples $(x, k) \subseteq \Sigma^* \times \mathbb{N}$, we call k the *(the value of the) parameter*.

Parameterized problems that can be decided by algorithms with runtime bounds of $O(f(k) \cdot p(|x|))$ for all inputs $(x, k) \in \Sigma^* \times \mathbb{N}$, where f is a computable function and p is a polynomial, are called *fixed-parameter tractable*. Such problems are collected in the class FPT. There exists an infinite W-hierarchy above FPT, $\mathrm{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots$, which is not known to be strict. Intuitively and informally, a PP shown to be W[1]-hard is, similar as for NP-hardness in the classical setting, considered to be a proof of intractability for that problem.
Given two PPs $\Pi, \Delta \subseteq \Sigma^* \times \mathbb{N}$, we say that Π *fpt-reduces* to Δ if (i) there exists an FPT-computable function f, such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ we have that $(x, k) \in \Pi$ if and only if $f(x, k) \in \Delta$; (ii) there exists a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ and $f(x, k) = (y, l)$ we have that $l \leq g(k)$.

Parameterized Complexity of Anti-Key. In this segment, we show that, from a parameterized perspective, determining the existence of an anti-key of a given cardinality is a W[1]-hard problem. This indicates that the problem is intractable. This sets expectations for the complexity of the interview process: with a current set of keys C, we need to identify the set R of meaningful minimal keys. To determine minimality, we need to interview the maximal anti-keys for C, that is, all those column sets that are maximal with the property of not currently being perceived as keys. First, we will provide some formal definitions required to state the problem.

Given a set of keys C over table schema T, a column set $A \subseteq T$ is an *anti-key* for C over T if and only if there is no $K \in C$ such that $K \subseteq A$. A known method for computing anti-keys is to take the complement of the minimum hitting sets of the hypergraph formed by the keys [24]. Now, define the corresponding decision problem.

Problem:	Anti-Key
Input:	Some set of keys C over table schema T, a natural number k.
Question:	Is there an anti-key A for C over T with cardinality $\lvert A\rvert = k$?

Theorem 4.1. Anti-Key *w.r.t. the parameter* k *is* W[1]*-hard.*

Proof. We show that the known W[1]-complete problem Clique [15, Thm. 21.2.4] fpt-reduces to Anti-Key. A graph $G = (V, E)$ and number $k \in \mathbb{N}$ are in Clique, if and only if G contains a clique of size k, i. e., there is a subset $V' \subseteq V$ of size $|V'| = k$ that forms a complete graph. Now, given an instance (G, k), construct an instance $f(G, k)$ of Anti-Key, by setting the table schema T to be the vertices V of the graph, the set of keys C to be the edges *not* in E, i. e., $C = \{\, \{u, v\} \mid u \neq v \text{ and } \{u, v\} \notin E \,\}$ and keeping the parameter k as is. Now, with the aforementioned connection between anti-keys and the complement of hitting sets (or vertex cover in this case), we have a one-to-one-correspondence between anti-keys and cliques. If G contains a clique of size k then $f(G, k)$ has an anti-key for C over T of the same size and *vice versa*. The function f is computable even in polynomial-time yielding W[1]-hardness as desired. □

Enumeration complexity. In this segment, we will study another kind of problems, namely, enumeration problems [29, 49]. In such cases, the aim is to print the full set of solutions for a given instance, avoiding any duplicates. Given an alphabet Σ, an *enumeration problem (EP)* is a tuple (I, Sol), where $I \subseteq \Sigma^*$ is the set of *instances*, $\mathrm{Sol}\colon I \to \mathcal{P}(\Sigma^*)$ is a function mapping each instance $x \in I$ to a finite set of *solutions*, and there exists a polynomial p such that for all $x \in I$ and for all $y \in \mathrm{Sol}(x)$, we have that $|y| \leq p(|x|)$ is true. As these problems typically involve an exponentially large solution space, they require exponential runtime algorithms. Now, we are ready to define the following two enumeration problems.

Problem:	ANTI-KEY-ENUM
Input:	A set of keys C over table schema T.
Output:	The set of anti-keys for C over T.

In line with the concept of Armstrong databases (see Section 2.2), we formulate the following enumeration problem.

Problem:	ARMSTRONG-ENUM
Input:	A set of keys C over table schema T.
Output:	List all records for some table over R that is Armstrong for C.

Similarly, let ARMSTRONG be the decision version of the previously defined enumeration problem.

Given a hypergraph $\mathcal{H} = (V, E)$, a *transversal* (or hitting set) $Tr \subseteq V$ is a set of vertices that intersects every hyperedge: $Tr \cap e \neq \emptyset$ for all $e \in E$. The *transversal hypergraph* $\mathrm{Tr}(\mathcal{H})$ consists of all inclusion minimal transversals of $\mathcal{H}$.

Problem:	HYPERGRAPH-TRANSVERSAL-ENUM
Input:	A hypergraph $\mathcal{H}$.
Output:	List all edges of $\mathrm{Tr}(\mathcal{H})$.

The previously defined problem is considered as the most important yardstick in enumeration complexity [4, 22]. In this area, the major open question is whether this problem can be solved in output-polynomial time (hence polynomial time bounded in both the input and output size). Let HYPERGRAPH-TRANSVERSAL be the decision problem, given two hypergraphs $\mathcal{H} = (V, E), \mathcal{G} = (V, F)$, to decide whether $\mathcal{G} = \mathrm{Tr}(\mathcal{H})$.

THEOREM 4.2. ANTI-KEY, HYPERGRAPH-TRANSVERSAL, *and* ARMSTRONG are equivalent under *p-m*-reductions that preserve the solution sizes.

PROOF. There is a one-to-one-correspondence between edges of the transversal hypergraph and the records in the Armstrong table [22, p. 3]. □

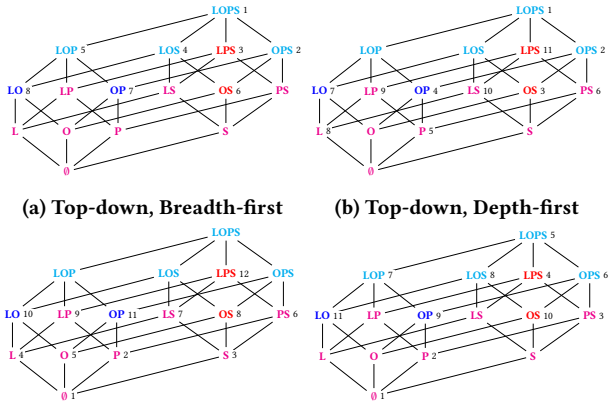
Hence, it is an open problem whether there is an output-polynomial algorithm for enumerating (i) the anti-keys for any given set C of keys currently perceived as meaningful, and (ii) the records of some table that is Armstrong for a given set C of minimal keys. These results provide clear expectations for the general tractability of the methodology illustrated in Fig. 3.

5 Algorithms

In this section, we will present our algorithmic contributions, starting with some overall outline and motivating example, and a description of the algorithms itself.

5.1 Overall Outline

We will present a suite of algorithms that implement fundamental strategies for traversing the Boolean powerset lattice of the underlying schema. Input to every algorithm is the table schema itself, a set of predetermined minimal keys resulting from previous analysis that is possibly empty, and a traversal strategy. The latter has a starting point, which is either the empty set or the entire column set determining whether it is a bottom-up or top-down approach, and a direction, which is either breadth-first or depth-first. The strategy determines the order in which column sets are put forward to a human expert team in form of a Yes/No question. "Yes" means that column set is an anti-key, while "No" means it is a key. The answer determines how the search space is pruned: In the first case, all subsets will be anti-keys needing no further questioning, and in the second case, all super-sets will be keys requiring no further questioning. Hence, the interview process with the human expert team is very much interactive.



(a) Top-down, Breadth-first **(b) Top-down, Depth-first**

(c) Bottom-up, Breadth-first **(d) Bottom-up, Depth-first**

Figure 5: Traversal strategies shown on running example

Building on Example 1.1 we are illustrating the four traversal strategies in Fig. 5. The numbers next to the column sets indicate the traversal order of candidate keys for which the Boolean questions are presented to the human experts. In this specific case, the distribution of the two minimal keys is best explored by the top-down, breadth-first strategy (TB) in Fig. 5a, which requires eight questions. Fig. 5b illustrates the top-down, depth-first strategy (TD) that needs 11 questions, Fig. 5c shows the bottom-up, breadth-first strategy (BB) that needs 12 questions, and Fig. 5d shows the bottom-up, depth-first strategy (BD) that needs 11 questions.

Based on Sec. 4, the enumeration process is worst-case exponential in the schema size, and it is an open problem whether an algorithm exists that is polynomial in the size of the output. Intuitively, this is the price for being assured that all candidates for minimal keys have been considered. Ultimately, which algorithm performs best depends largely on the distribution of the output. Experiments will investigate this question and others in Sec. 6.

5.2 Algorithm Suite

Our algorithms comprise Algs. 1-4, with Alg. 4 being the main algorithm. Algs. 1 and 2 implement bottom-up and top-down approaches, while Alg. 3 initializes the interview questions.

In Alg. 1 (Alg. 2, respectively), line 1 recognizes super keys with given rule set in current round. line 2 recognizes anti-keys with anti-keys in current round. Then it removes super keys and anti-keys from candidates that have been considered at line 3. Line 4 initializes rule sets and anti-keys for current round. Lines 5–9 extend subsets of anti-keys by adding suitable columns (reduce superkeys by removing suitable columns, respectively) to generate new candidates. Afterwards, the algorithm stores candidates which are

Algorithm 1 updateCandSetBtmUp(T, R_c, C, A_c, A, R, R_{in})

Require: T: Table schema, R_c: Rule set in the round, C: Candidate set for interview, A_c: Set of anti-keys in the round, A: Set of anti-keys, R: Rule set, R_{in}: Input set of minimal keys.
Ensure: Updated C

1: $SKs \leftarrow \{S \in C \mid K \subseteq S,\ \forall K \in R_c\}$
2: $AKs \leftarrow \{S \in C \mid S \subseteq NK,\ \forall NK \in A_c\}$
3: $C \leftarrow C \setminus (SKs \cup AKs)$ ▷ remove super keys & anti-keys
4: $R_c \leftarrow \emptyset, A_c \leftarrow \emptyset$
5: **for each** $AK \in AKs$ **do**
6: **for each** $e \in (T \setminus AK)$ **do** ▷ extend anti-keys
7: $CK \leftarrow AK \cup \{e\}$
8: **if** $r \not\subseteq CK, \forall r \in R\ \&\ CK \not\subseteq a, \forall a \in A \cup R_{\text{in}}$ **then**
9: $C \leftarrow C \cup \{CK\}$
10: **if** $CK \subset a, \exists a \in A \cup R_{\text{in}}$ **then**
11: $A_c \leftarrow A_c \cup \{CK\}$
12: **return** decreasing order of C

Algorithm 2 updateCandSetTopDown(R_c, C, A_c, A, R, R_{in})

Require: R_c: Rule set in the round, C: Candidate set for interview, A_c: Set of anti-keys in the round, A: Set of anti-keys, R: Rule set, R_{in}: Input set of minimal keys.
Ensure: Updated C

1: $SKs \leftarrow \{S \in C \mid K \subseteq S,\ \forall K \in R_c\}$
2: $AKs \leftarrow \{S \in C \mid S \subseteq NK,\ \forall NK \in A_c\}$
3: $C \leftarrow C \setminus (SKs \cup AKs)$ ▷ remove super keys & anti-keys
4: $R_c \leftarrow \emptyset, A_c \leftarrow \emptyset$
5: **for each** $SK \in SKs$ **do**
6: **for each** $e \in SK$ **do** ▷ refine super keys
7: $CK \leftarrow SK \setminus \{e\}$
8: **if** $r \not\subseteq CK, \forall r \in R\ \&\ CK \not\subseteq a, \forall a \in A \cup R_{\text{in}}$ **then**
9: $C \leftarrow C \cup \{CK\}$
10: **if** $r \subset CK, \exists r \in R$ **then**
11: $R_c \leftarrow R_c \cup \{CK\}$
12: **return** increasing order of C

anti-keys (super keys) at lines 10-11. Finally, the algorithm returns the candidate in decreasing (increasing, respectively) order.
In Alg. 3, lines 1–2 return the original input when no predetermined keys are given. Line 3 initializes all sets. Lines 4–12 generate new candidates based on the traversal direction: for top-down, suitable columns are removed; for bottom-up, suitable columns are added to obtain new candidates. Lines 13–14 are recursive calls to explore the next layer when no new candidates have been found.

In Alg. 4, Lines 1–2 initialize necessary variables. Line 3 initializes the question set for interview, using Alg. 3. Lines 5–13 comprise the interview for the acquisition of minimal keys. For top-down (bottom-up) depth-first, the interview loop exits after finding a key (anti-key), while for breadth-first, the interview traverses all candidates. Lines 15–18 update candidates for the next round by applying Alg. 1 and Alg. 2.

The algorithms ensure that a minimum number of column sets are enumerated to identify the set of meaningful minimal keys under the given traversal strategy. The output of Alg. 4 is the set of meaningful minimal keys, as perceived by the human expert team with respect to the answers they provide.

Algorithm 3 initCandSet(T, S_{trav}, R, A_c, R_c, C_i)

Require: T: Table schema, S_{trav}: Traversal start, R: Rule set, A_c: anti-keys in the round, R_c: Rule set in the round, C_i: Initial set.
Ensure: Initial candidate sets C, rule set R_c in the round, and anti-keys A_c in the round.

1: **if** $R = \emptyset$ **then**
2: **return** C_i, R_c, A_c
3: Initialize sets $C \leftarrow \emptyset, C_{\text{all}} \leftarrow \emptyset, R_c \leftarrow \emptyset, A_c \leftarrow \emptyset$
4: **for each** $X \in C_i$ **do**
5: **if** S_{trav} = top-down **then**
6: $CKs \leftarrow \{X \setminus \{a\} \mid a \in X\}$
7: $R_c \leftarrow R_c \cup \{CK \in CKs \mid r \subset CK,\ \exists r \in R\}$
8: **else**
9: $CKs \leftarrow \{X \cup \{a\} \mid a \in T \setminus X\}$
10: $A_c \leftarrow A_c \cup \{CK \in CKs \mid CK \subset r,\ \exists r \in R\}$
11: $C \leftarrow C \cup \{CK \in CKs \mid r \not\subseteq CK\ \&\ CK \not\subseteq r,\ \forall r \in R\}$
12: $C_{\text{all}} \leftarrow C_{\text{all}} \cup CKs$
13: **if** $C = \emptyset$ **then**
14: **return** initCandSet(T, S_{trav}, R, A_c, R_c, C_{all})
15: **return** C, R_c, A_c

Algorithm 4 Key Interview Framework

Require: S_{trav}: Traversal start, D_{trav}: Traversal direction, R_{in}: Input set of minimal keys, T: Table schema.
Ensure: Rule set R acquired from interview.

1: Initialize sets: $C \leftarrow \emptyset, R \leftarrow R_{\text{in}}, A \leftarrow \emptyset, R_c \leftarrow \emptyset, A_c \leftarrow \emptyset$
2: $C_i \leftarrow \{T\}$ for top-down, otherwise $C_i \leftarrow \{\emptyset\}$
3: $C, R_c, A_c \leftarrow$ initCandSet(T, S_{trav}, R, A_c, R_c, C_i)
4: **while** $C \neq \emptyset$ *or* $A_c \neq \emptyset$ *or* $R_c \neq \emptyset$ **do**
5: **for each** $X \in C$ **do**
6: **if** X is a key **then**
7: $R_c \leftarrow R_c \cup \{X\}, R \leftarrow R \cup \{X\}$
8: **if** S_{trav} = topdown & D_{trav} = dfs **then**
9: **goto** ExitLoops
10: **else**
11: $A_c \leftarrow A_c \cup \{X\}, A \leftarrow A \cup \{X\}$
12: **if** S_{trav} = bottomup & D_{trav} = dfs **then**
13: **goto** ExitLoops
14: **flag**: ExitLoops
15: **if** S_{trav} = top-down **then**
16: $C \leftarrow$ updateCandSetTopDown($R_c, C, A_c, A, R, R_{\text{in}}$)
17: **else**
18: $C \leftarrow$ updateCandSetBtmUp($T, R_c, C, A_c, A, R, R_{\text{in}}$)
19: **return** R

6 Experiments

We conduct several experiments to provide insight into the acquisition process of database keys we have proposed.

6.1 Setup and Availability

All experiments were implemented in Java 17 and executed on a server running Windows 11. The hardware configuration consists of a 12th Gen Intel(R) Core(TM) i7-12700 CPU @ 2.10 GHz and 128 GB of RAM. The source code, interview tool, datasets and materials are available at *https://anonymous.4open.science/r/interview4key*. Our interview tool is ready for use.

6.2 Research Questions

In view of the worst-case intractability of the enumeration process, our experiments quantify how well the traversals strategies perform and how well they compare to one another when run on synthetic and real-world distributions of minimal keys. We will investigate the following research questions. **RQ1**: How does the distribution of minimal keys affect the performance of our algorithms? **RQ2**: How does the schema size affect the performance of our algorithms? **RQ3**: How well do the algorithms perform for distributions of minimal keys mined from real-world data sets **RQ4:** How well do the algorithms perform for distributions of real-world minimal keys? **RQ5:** How much do predetermined minimal keys improve the efficiency of our algorithms? **RQ6:** How much can we improve the efficiency of our algorithms by removing columns from the schema that are unlikely to be part of any minimal key?

6.3 Impact of Minimal Key Distribution

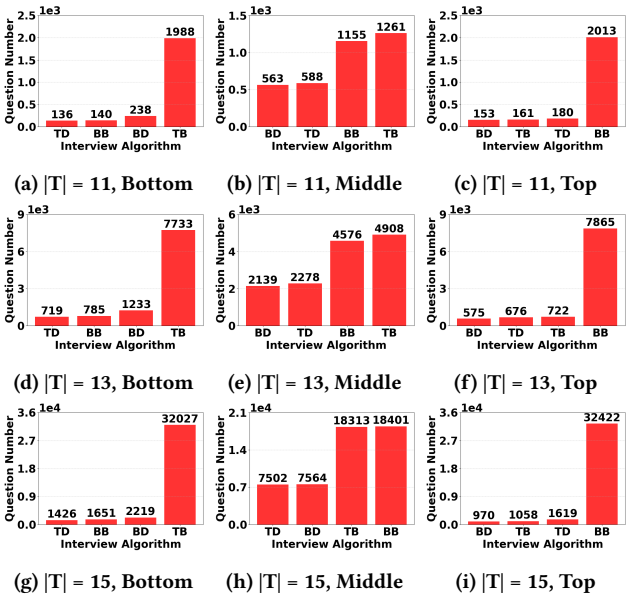
This subsection investigates the first research question **RQ1**. It aims at identifying how well the traversal strategies work when minimal keys are distributed according to general characteristics of the schema's lattice. All experiments are performed over schemata of size 11, 13, and 15. In each setting, we perform 100 different runs to ensure we capture a diverse range of distributions with a particular characteristic under consideration.

6.3.1 Random Distribution of Minimal Keys. As a first experiment, we consider the general case where the set of minimal keys is distributed randomly across any layers in the powerset lattice.

Fig. 6 shows the results of our experiments when we take the average size k across all minimal keys from a given distribution to determine to which type of layer the experiment results belong. Here, the layers are classified into three types: the Bottom layers are those where $k \in [0, \frac{n}{3}]$, the Middle layers identify as those where $k \in (\frac{n}{3}, \frac{2n}{3}]$, and Top layers as those where $k \in (\frac{2n}{3}, n]$.

The results show what one would expect, but quantify these expectations. For bottom layers, top-down breadth-first is prohibitively expensive; and dually, for top layers, bottom-up breadth-first is prohibitively expensive. The other algorithms perform comparatively well on bottom and top layers. The middle layers are the most expensive by far, and breadth-first traversals are expensive for both bottom-up and top-down strategies, while depth-first strategies perform significantly better. Having minimal keys across different layers requires traversal of these layers but also offers opportunities for pruning some supersets of keys and subsets of anti-keys.

Apart from considering the average size of minimal keys across a given distribution, we also consider the *boundary* of the distribution: the difference between the maximum and minimum size for minimal keys within a given distribution. If this is zero, all minimal

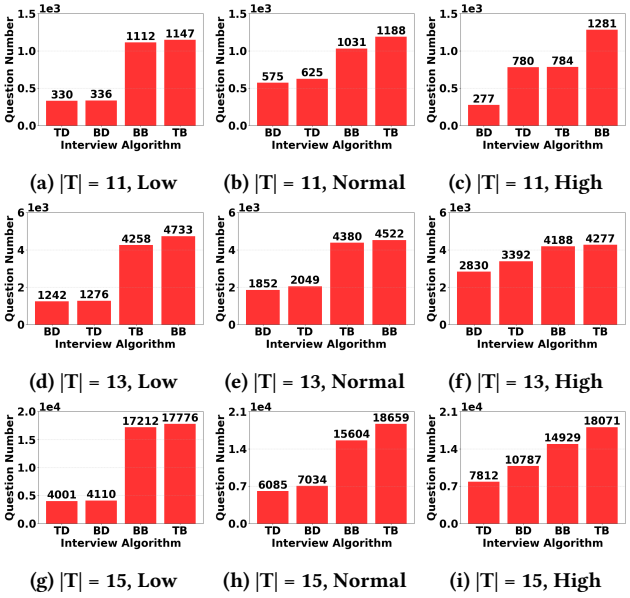


(a) |T| = 11, Bottom (b) |T| = 11, Middle (c) |T| = 11, Top

(d) |T| = 13, Bottom (e) |T| = 13, Middle (f) |T| = 13, Top

(g) |T| = 15, Bottom (h) |T| = 15, Middle (i) |T| = 15, Top

Figure 6: Number of questions required for random distributions of minimal keys across all lattice layers on a schema with n columns, classified by the average size k of minimal keys in the distribution. Depending to which range k belongs, we distinguish between Bottom ($[0, \frac{n}{3}]$), Middle ($(\frac{n}{3}, \frac{2n}{3}]$) and Top ($(\frac{2n}{3}, n]$) layers.

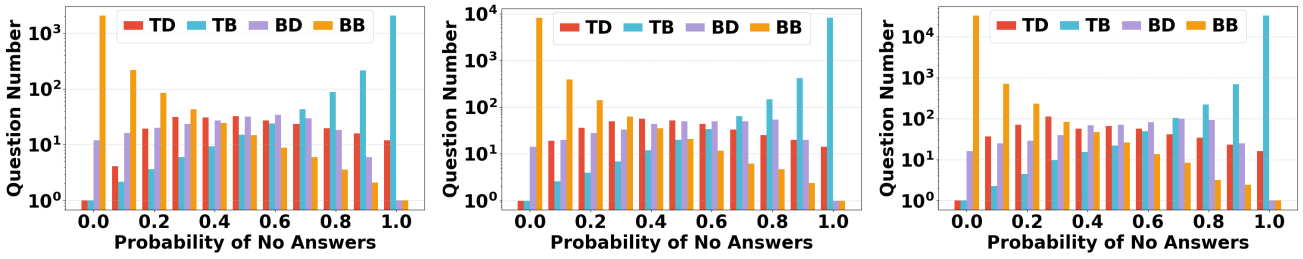


(a) |T| = 11, Low (b) |T| = 11, Normal (c) |T| = 11, High

(d) |T| = 13, Low (e) |T| = 13, Normal (f) |T| = 13, High

(g) |T| = 15, Low (h) |T| = 15, Normal (i) |T| = 15, High

Figure 7: Number of questions required for random distributions of minimal keys across all lattice layers on a schema with n columns, classified by the distance between the maximal and minimal size of minimal keys that occur within the distribution: low ($[0, \frac{n-2}{3}]$), normal ($(\frac{n-2}{3}, \frac{2n-4}{3}]$), and high ($(\frac{2n-4}{3}, n-2]$).



(a) |T| = 11 (b) |T| = 13 (c) |T| = 15

Figure 8: Interview question number across all interview algorithms under probabilistic answering.

keys occur in the same layer. If there is some minimal key with one column and another minimal key with $n-1$ columns, the difference is $n-2$, which is the maximum boundary. Similar to the bottom, middle, and top layers from before, we distinguish between the following ranges of boundaries: Low ($[0, \frac{n-2}{3}]$), Normal ($(\frac{n-2}{3}, \frac{2n-4}{3}]$), and High ($(\frac{2n-4}{3}, n-2]$).

Fig. 7 quantifies the impact of the boundary on the performance of our algorithms. On low boundaries, the depth-first strategies outperform the breadth-first strategies. This becomes less pronounced but still significant when the boundary grows. Intuitively, the larger boundaries are, the more depth-first traversals are required to find all minimal keys. The preference for bottom-up or top-down depends on details of the distribution, that is, how quickly anti-keys and keys can be used for pruning.

6.3.2 Probabilistic Distributions of Minimal Keys. Our previous experiments have focused on the distribution of the minimal keys across the underlying lattice structure. We will now investigate a different angle where we vary the density of minimal keys. In other words, how does the number of minimal keys impact the performance of our algorithms? During the interviews we identify a key whenever the answer to a question is 'No'. Consequently, we define the probability p by which an answer is 'No' during an interview. We then conduct experiments by running our algorithms that answer 'No' with probability p for $p = 0, 0.1, \ldots, 0.9, 1$.

Fig. 8 shows the results for these experiments on the three different schema sizes. The figures show remarkable insight in terms of the symmetry how well the different traversal strategies perform.

With no key present ($p = 0$), the top-down strategies identify the full schema immediately as an anti-key, hence only one question is required. BB needs to traverse all layers of the lattice from bottom to top to eventually realize that no key exists, while BD traverses up the lattice along a path from bottom to top. These observations reverse when all column sets are keys ($p = 1$). In this case, the empty set is the only minimal key, and the bottom-up algorithms identify it with one question. TB traverses all column sets from top to bottom until it reaches the empty set eventually, while TD traverses the lattice from top to bottom along a path.

For the breadth-first traversals TB and BB, Fig. 8 shows their symmetry in generating interview questions: the higher p, the more (less) efficient BB (TB) becomes, with a tie at $p = 0.5$.

Fig. 8 also shows strong symmetry between the depth-first traversals TD and BD: TD and BD become less efficient when p grows from 0 to $0.5 - 0.6$, and then become more efficient when p grows further. TD outperforms BD at 0 and 0.1, they are tied at 0.2 and 0.8, while BD outperforms TD at 0.9 and 1. Within $0.3 - 0.7$ they show similar performance with BD being slightly better than TD at 0.3 and 0.4, and TD being slightly better than BD at 0.6 and 0.7.

TB is the best performer within the low range of probabilities $[0, 0.5)$: with fewer minimal keys present, answering 'Yes' to column sets of large size identifies them as (maximal) anti-keys which results in fast pruning. Dually, BB is the best performer within the high range of probabilities $(0.5, 1]$: answering 'No' to column sets of low size identifies them as (minimal) keys which results in fast pruning. These observations hold consistently across the different schema sizes we have examined.

6.4 Impact of Schema Size

An important parameter for the efficiency of any interview algorithm is the number n of columns in the underlying schema. Ultimately, the search space has 2^n candidates, and even the maximum solution space has size $\binom{n}{\lfloor n/2 \rfloor}$, which is exponential in n.

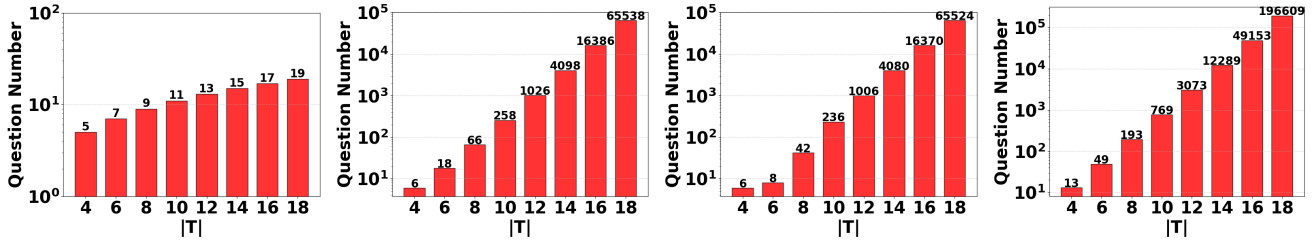
For **RQ2**, we analyze a simple medical scenario in which the schema T_n consists of the $n+2$ columns: $I, D, S_1, \ldots, S_n, O_1, \ldots, O_n$ where I denotes the illness, D the diagnostic test result, S_i represents a symptom, and O_i corresponds to a treatment option. For each choice of n, $\mathcal{S}_n$ has only the minimal key $\{I, D\}$.

In the experiments for **RQ2**, we set n from 1 to 8 to synthesize 8 schemata with 4 to 18 columns. As illustrated in Fig. 9, most algorithms exhibit a consistent trend, with question numbers increasing sharply with n that quantifies the expected exponential behavior. An exception is the TD algorithm that is capable of finding the minimal key through an efficient pruning of anti-keys on the top and identifying the minimal key by depth-first traversal.

On the one hand, the results highlight the combinatorial explosion of the search space in the schema size. On the other hand, they also show an opportunity for a practical approach: If design teams can identify columns that are unlikely to occur in any minimal key, then the search space can be reduced significantly as well. We will investigate this opportunity later.

6.5 Real-world Scenarios

Our previous experiments have surfaced the advantages and disadvantages for each of our algorithms, providing justifications for them with general distributions of minimal keys. We will now



(a) TD (b) TB (c) BD (d) BB

Figure 9: Interview question number across all interview algorithms under schemata $\mathcal{S}_n$.

Table 1: Statistics for Interviews about Sets of Minimal Keys Mined from Real-world Datasets

name	$\|T\|$	$\|R \setminus R_{\text{in}}\|$	$R \setminus R_{\text{in}}$ *distr*	$\|R_{\text{in}}\|$	R_{in} *distr*	*Ranked #Q with p*
abalone	9	29	3, 4.45, 6 (middle, normal)	-	-	BD : 199, TD : 211, TB : 252, BB : 319 BD : 0.82, TD : 0.17, TB : 0.88, BB : 0.09
		20	4, 4.75, 6 (middle, low)	9	3, 3.78, 4 (middle, low)	BD : 81, TB : 84, TD : 167, BB : 256 BD : 0.58, TB : 0.64, TD : 0.14, BB : 0.08
	4	1	3, 3.00, 3 (top, low)	-	-	TB : 5, TD : 6, BD : 8, BB : 15 TB : 0.4, TD : 0.33, BD : 0.25, BB : 0.07
	-	-	-	-	-	-
breast	11	2	5, 5.00, 5 (middle, low)	-	-	TB : 101, BD : 104, TD : 225, BB : 1954 TB : 0.95, BD : 0.86, TD : 0.04, BB : 0
		1	5, 5.00, 5 (middle, low)	1	5, 5.00, 5 (middle, low)	TB : 37, TD : 39, BD : 46, BB : 1922 TB : 0.86, TD : 0.15, BD : 0.7, BB : 0
	5	1	5, 5.00, 5 (top, low)	-	-	TD : 6, TB : 6, BD : 10, BB : 32 TD : 0.17, TB : 0.17, BD : 0.1, BB : 0.03
	-	-	-	-	-	-
echo	13	39	3, 3.85, 4 (bottom, low)	-	-	TB : 3497, BD : 3509, TD : 4105, BB : 4749 TB : 1, BD : 0.99, TD : 0.01, BB : 0.01
		26	4, 4.00, 4 (bottom, low)	13	3, 3.54, 4 (bottom, low)	TB : 449, BD : 460, TD : 568, BB : 4669 TB : 0.97, BD : 0.94, TD : 0.06, BB : 0.01
	6	4	4, 4.00, 4 (middle, low)	-	-	TB : 14, BD : 19, TD : 27, BB : 59 TB : 0.64, BD : 0.47, TD : 0.22, BB : 0.07
		2	4, 4.00, 4 (middle, low)	2	4, 4.00, 4 (middle, low)	TB : 7, BD : 11, TD : 18, BB : 35 TB : 0.43, BD : 0.27, TD : 0.17, BB : 0.06
claims	13	1	1, 1.00, 1 (bottom, low)	-	-	TD : 14, BD : 3086, BB : 4097, TB : 4097 TD : 0.93, BD : 1, BB : 0, TB : 1
		-	-			-
	6	1	1, 1.00, 1 (bottom, low)			BD : 7, TD : 7, BB : 33, TB : 33 BD : 0.14, TD : 0.86, BB : 0.03, TB : 0.97
	-	-	-			-
hospital	15	12	2, 2.92, 4 (bottom, low)	-	-	TD : 10249, BB : 12364, BD : 15980, TB : 20420 TD : 0, BB : 0, BD : 1, TB : 1
		8	3, 3.25, 4 (bottom, low)	4	2, 2.25, 3 (bottom, low)	BD : 3884, TB : 4036, TD : 10840, BB : 12350 BD : 1, TB : 1, TD : 0, BB : 0
	7	7	2, 2.71, 3 (middle, low)	-	-	BD : 40, TD : 55, BB : 67, TB : 72 BD : 0.75, TD : 0.22, BB : 0.1, TB : 0.94
		4	3, 3.00, 3 (middle, low)	3	2, 2.33, 3 (bottom, low)	TD : 10, TB : 16, BD : 17, BB : 55 TD : 0.6, TB : 0.75, BD : 0.47, BB : 0.07
weather	18	523	2, 7.29, 10 (middle, normal)	-	-	TD : 105868, BD : 115871, BB : 122295, TB : 140602 TD : 0.01, BD : 1, BB : 0, TB : 1
		393	7, 7.85, 10 (middle, low)	130	2, 5.62, 7 (bottom, low)	BD : 8791, TB : 8802, TD : 38863, BB : 120128 BD : 0.97, TB : 0.97, TD : 0.01, BB : 0
	9	2	2, 3.00, 4 (bottom, low)	-	-	BD : 147, TB : 148, TD : 187, BB : 370 BD : 0.92, TB : 0.97, TD : 0.05, BB : 0.01
		1	4, 4.00, 4 (middle, low)	1	2, 2.00, 2 (bottom, low)	TB : 20, BD : 27, TD : 126, BB : 366 TB : 0.8, BD : 0.59, TD : 0.04, BB : 0
routes	9	2	3, 3.50, 4 (middle, low)	-	-	BD : 77, TB : 84, TD : 161, BB : 434 BD : 0.84, TB : 0.95, TD : 0.05, BB : 0
		1	4, 4.00, 4 (middle, low)	1	3, 3.00, 3 (bottom, low)	TB : 20, BD : 27, TD : 138, BB : 426 TB : 0.8, BD : 0.59, TD : 0.04, BB : 0
	4	1	3, 3.00, 3 (top, low)	-	-	TB : 5, BD : 7, TD : 8, BB : 15 TB : 0.4, BD : 0.14, TD : 0.25, BB : 0.07
	-	-	-	-	-	-
bridges	13	3	1, 1.67, 2 (bottom, low)	-	-	TD : 2417, BB : 2563, BD : 3343, TB : 5634 TD : 0.01, BB : 0, BD : 1, TB : 1
		2	2, 2.00, 2 (bottom, low)	1	1, 1.00, 1 (bottom, low)	BD : 1421, TB : 1538, BB : 2561, TD : 2568 BD : 0.99, TB : 1, BB : 0, TD : 0
	6	3	1, 1.67, 2 (bottom, low)	-	-	BD : 16, BB : 23, TD : 25, TB : 46 BD : 0.63, BB : 0.13, TD : 0.32, TB : 0.96
		2	2, 2.00, 2 (bottom, low)	1	1, 1.00, 1 (bottom, low)	BD : 14, TB : 14, BB : 21, TD : 21 BD : 0.64, TB : 0.86, BB : 0.1, TD : 0.24
pdbx	13	11	1, 3.55, 5 (bottom, normal)	-	-	TD : 2260, BB : 2799, BD : 3340, TB : 5409 TD : 0.01, BB : 0, BD : 1, TB : 1
		8	3, 4.00, 5 (bottom, low)	3	1, 2.33, 3 (bottom, low)	BD : 520, TB : 545, TD : 751, BB : 2786 BD : 0.97, TB : 0.99, TD : 0.02, BB : 0
	6	2	1, 2.00, 3 (bottom, low)	-	-	BD : 9, TD : 10, BB : 30, TB : 39 BD : 0.22, TD : 0.7, BB : 0.07, TB : 0.92
		1	3, 3.00, 3 (middle, low)	1	1, 1.00, 1 (bottom, low)	TD : 6, TB : 7, BD : 9, BB : 28 TD : 0.5, TB : 0.57, BD : 0.33, BB : 0.04
adult	15	2	10, 10.00, 10 (middle, low)	-	-	TB : 58, BD : 72, TD : 277, BB : 32722 TB : 0.83, BD : 0.67, TD : 0.03, BB : 0
		1	10, 10.00, 10 (middle, low)	1	10, 10.00, 10 (middle, low)	TB : 26, BD : 39, TD : 224, BB : 31698 TB : 0.62, BD : 0.41, TD : 0.02, BB : 0
	10	1	10, 10.00, 10 (top, low)	-	-	TD : 11, TB : 11, BD : 20, BB : 1024 TD : 0.09, TB : 0.09, BD : 0.05, BB : 0
	-	-	-	-	-	-
lineitem	16	390	2, 5.47, 8 (middle, normal)	-	-	TD : 11986, BB : 13161, BD : 24492, TB : 53036 TD : 0.03, BB : 0.03, BD : 0.99, TB : 0.99
		293	5, 5.97, 8 (middle, low)	97	2, 3.97, 5 (bottom, low)	BD : 4033, TB : 4244, TD : 11313, BB : 12668 BD : 0.93, TB : 0.94, TD : 0.03, BB : 0.02
	8	6	3, 4.17, 5 (middle, low)	-	-	BD : 59, TB : 62, TD : 101, BB : 208 BD : 0.75, TB : 0.87, TD : 0.11, BB : 0.03
		3	4, 4.67, 5 (middle, low)	3	3, 3.67, 4 (middle, low)	TB : 18, BD : 24, TD : 33, BB : 180 TB : 0.56, BD : 0.42, TD : 0.15, BB : 0.02
ncvoter	19	113	3, 4.53, 7 (bottom, low)	-	-	TD : 232701, BD : 249060, TB : 252281, BB : 272163 TD : 0, BD : 1, TB : 1, BB : 0
		85	4, 4.91, 7 (bottom, low)	28	3, 3.39, 4 (bottom, low)	TD : 5859, TB : 7865, BD : 7878, BB : 272027 TD : 0.02, TB : 0.99, BD : 0.99, BB : 0
	9	2	3, 3.00, 3 (bottom, low)	-	-	BD : 96, TB : 99, TD : 135, BB : 418 BD : 0.89, TB : 0.97, TD : 0.06, BB : 0
		1	3, 3.00, 3 (bottom, low)	1	3, 3.00, 3 (bottom, low)	TB : 35, BD : 39, TD : 58, BB : 410 TB : 0.91, BD : 0.74, TD : 0.1, BB : 0

address research questions **RQ3**-**RQ6**, for which we explore different real-world scenarios and some general pragmatic approaches towards reducing computational overheads.

6.5.1 Minimal Keys Mined from Real-world Data. Our first experiment on real-world scenarios uses the set of minimal keys mined from twelve real-world datasets, addressing **RQ3**. Not only do we acknowledge that not all mined keys are meaningful, but our main contribution is dedicated towards addressing this point. Nevertheless, our experiments show which efforts each of our traversal strategies require in obtaining these keys from an interview process. In other words, we answer 'No' to the column set in question whenever it is a superkey for some minimal key that was mined.

Table 1 presents the statistics for all datasets. This includes the schema size ($|T|$), the number $|R_{\text{in}}|$ of minimal keys from R we use as predetermined keys, the number ($|R \setminus R_{\text{in}}|$) of minimal keys without predetermined ones, their distributions with the minimum, average, and maximum layers across the minimal keys of a distribution, their types of regions and boundaries, and the ranking of algorithms based on the number of interview questions required (Ranked #Q with p), and the percentage of 'No' answers.
Minimal keys are mostly few with some exceptions, and these occur mostly in the bottom, sometimes in the middle and rarely in the top layer. Boundaries are mostly low and sometimes normal.

Apart from BB, there are settings in which each of the other algorithms performs best. TD works well as it combines (i) finding

Table 2: Statistics for Interviews about Sets of Real-world Minimal Keys

name	$\|T\|$	$\|R \setminus R_{\text{in}}\|$	$R \setminus R_{\text{in}}$ *distr*	$\|R_{\text{in}}\|$	R_{in} *distr*	*Ranked #Q with p*
TeamsPost	17	1	2, 2.00, 2 (bottom, low)	-	-	TD : 18, BD : 24595, TB : 32770, BB : 98305 TD : 0.89, BD : 1, TB : 1, BB : 0
		-	-	-	-	-
	3	1	2, 2.00, 2 (middle, low)	-	-	TD : 4, TB : 4, BD : 6, BB : 7 TD : 0.5, TB : 0.5, BD : 0.33, BB : 0.14
		-	-	-	-	-
lineitem-0	16	6	2, 2.67, 3 (bottom, low)	-	-	TD : 4370, BD : 28307, TB : 28419, BB : 37126 TD : 0, BD : 1, TB : 1, BB : 0
		3	3, 3.00, 3 (bottom, low)	3	2, 2.33, 3 (bottom, low)	TD : 268, TB : 1795, BD : 1809, BB : 37114 TD : 0.05, TB : 1, BD : 0.99, BB : 0
	8	6	2, 2.67, 3 (bottom, low)	-	-	TD : 27, BD : 106, TB : 114, BB : 151 TD : 0.44, BD : 0.91, TB : 0.97, BB : 0.04
		3	3, 3.00, 3 (middle, low)	3	2, 2.33, 3 (bottom, low)	TD : 7, TB : 9, BD : 15, BB : 139 TD : 0.71, TB : 0.78, BD : 0.47, BB : 0.02
TeamsHalf	11	2	3, 3.00, 3 (bottom, low)	-	-	BD : 334, TB : 387, TD : 388, BB : 1666 BD : 0.96, TB : 0.99, TD : 0.03, BB : 0
		1		1	3, 3.00, 3 (bottom, low)	TB : 131, BD : 140, TD : 204, BB : 1658 TB : 0.98, BD : 0.91, TD : 0.04, BB : 0
	5	2	3, 3.00, 3 (middle, low)	-	-	TB : 9, TD : 11, BD : 12, BB : 28 TB : 0.67, TD : 0.36, BD : 0.42, BB : 0.07
		1		1	3, 3.00, 3 (middle, low)	TB : 5, BD : 8, TD : 8, BB : 20 TB : 0.4, BD : 0.25, TD : 0.25, BB : 0.05
AwardsMisc	6	2	1, 1.00, 1 (bottom, low)	-	-	BB : 18, TD : 23, BD : 37, TB : 49 BB : 0.11, TD : 0.3, BD : 0.86, TB : 0.98
		1		1	1, 1.00, 1 (bottom, low)	BD : 5, BB : 16, TB : 17, TD : 18 BD : 0.2, BB : 0.06, TB : 0.94, TD : 0.28
	4	2		-	-	BB : 6, TD : 9, BD : 11, TB : 13 BB : 0.33, TD : 0.56, BD : 0.73, TB : 0.92
		1		1	1, 1.00, 1 (bottom, low)	BD : 3, BB : 4, TB : 5, TD : 6 BD : 0.33, BB : 0.25, TB : 0.8, TD : 0.5
lineitem-1	16	2	2, 2.00, 2 (bottom, low)	-	-	TD : 8206, BD : 24466, TB : 24578, BB : 40962 TD : 0, BD : 1, TB : 1, BB : 0
		1		1	2, 2.00, 2 (bottom, low)	TD : 16, BD : 8145, TB : 8194, BB : 40958 TD : 0.88, BD : 1, TB : 1, BB : 0
	4	2	2, 2.00, 2 (middle, low)	-	-	TD : 6, BD : 8, TB : 8, BB : 12 TD : 0.67, BD : 0.38, TB : 0.75, BB : 0.17
		1		1	2, 2.00, 2 (middle, low)	TD : 4, TB : 4, BD : 5, BB : 8 TD : 0.5, TB : 0.5, BD : 0.2, BB : 0.13
SeriesPost	13	4	2, 2.75, 3 (bottom, low)	-	-	TD : 1020, BD : 3064, TB : 3076, BB : 5124 TD : 0.01, BD : 0.99, TB : 1, BB : 0
		2	2, 2.50, 3 (bottom, low)	2	3, 3.00, 3 (bottom, low)	TD : 895, TB : 1540, BD : 1551, BB : 5112 TD : 0.01, TB : 1, BD : 0.99, BB : 0
	5	4	2, 2.75, 3 (middle, low)	-	-	TD : 11, BD : 16, TB : 16, BB : 24 TD : 0.64, BD : 0.5, TB : 0.75, BB : 0.17
		2	2, 2.50, 3 (middle, low)	2	3, 3.00, 3 (middle, low)	TD : 5, BD : 6, TB : 7, BB : 12 TD : 0.8, BD : 0.33, TB : 0.86, BB : 0.17

many irrelevant columns in large anti-keys at top layers with (ii) finding few minimal keys even in bottom layers through detecting large keys early at the top. TD is the most preferred strategy on larger schemata. BD works well as it combines (i) finding few minimal keys in bottom layers with (ii) pruning anti-keys as upward paths are traversed. BD is very efficient when the schema is smaller. While not featuring frequently as the best strategy, TB works well when multiple keys co-exist within boundaries close to zero (`breast`, `echo`, `adult`). TB can quickly find few keys or prune anti-keys in top layers, and confirm minimality of keys within close boundary quickly. BB does not work that well: While it may find small minimal keys quickly in lower layers, it still takes too long to find maximal anti-keys and prune effectively.

Regarding **RQ5** and **RQ6**, if the number of questions exceeds a hundred, this may be considered too exhaustive. However, adding minimal keys predetermined by other methods, removing columns that are unlikely to feature in any key, and combining these two techniques can make the interview process very feasible, as demonstrated by the numbers in the experiments here.

6.5.2 Real-world Minimal Keys. In addressing **RQ4**, **RQ5**, and **RQ6**, we conduct experiments on schemata from the Hockey database[1] and TPC-H[2], for which meaningful minimal keys are provided or can be extracted with confidence. The schema information and related experiment results are listed in Table 2. These real-world keys occur mostly at bottom and rarely at middle layers, exclusively with low boundaries. As before, we illustrate the impact of removing columns from the schema that are unlikely to occur in any meaningful minimal key, such as *l_comment* in `lineitem`, and also providing some of those minimal keys as part of the input. Noticeably, each of the algorithms performs the best in some scenario. As before, TD is preferred on larger schemata like `TeamsPost`, `lineitem`, and `SeriesPost` with 15 columns or more, or reduced schemata with minimal keys that have non-zero boundary like `lineitem-0` (8 columns) and `SeriesPost` (5). BD is preferred on schemata of moderate size with few keys at the bottom like `TeamsHalf` (11) with 2 keys or smaller schemata with few keys pre-determined in the same layer at the bottom like `AwardsMisc` (6 & 4) with one key. TB works well on reduced schemata where few minimal keys are located in middle layers with near-zero boundary like `TeamHalf` with 5 columns. Even BB has a real-world scenario in which multiple minimal keys co-occur in the same bottom layer of a small schema, in particular `AwardsDisc` with 6 and 4 columns, respectively. In this scenario, the location favors a bottom-up approach and depth-first explores multiple paths upwards before finding all the minimal keys located at the bottom, while breadth-first finds them quickly and, due to the small schema, can confirm the maximal anti-keys quickly due to the pruning by the minimal keys.

[1]https://relational.fel.cvut.cz/dataset/Hockey
[2]https://www.tpc.org/tpch/

7 Conclusion and Future Work

Identifying the set of meaningful minimal keys with perfect precision and recall is a prerequisite for correct data modeling, data integrity, and efficient data management. We have proposed a suite of algorithms that generates Boolean questions to a team of human experts which will lead to perfect precision and recall with the least number of questions required under the given strategy of traversal. Experiments with synthetic and real-world sets of minimal keys showcase how well the algorithms perform for different distributions of the keys. Removing columns from the search space that are unlikely to occur in any key and providing predetermined keys to the interview process are practical approaches to make the efficient acquisition of all minimal keys feasible, even under the strong intractability results we have shown to hold.

In future work, we will work on more efficient traversal strategies but also the modification of Armstrong relations as a complementary technique for the acquisition process.

References

[1] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. 2018. *Data Profiling.* Morgan & Claypool Publishers. doi:10.2200/S00878ED1V01Y201810DTM052

[2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savkovic, Michael Schmidt, Juan F. Sequeda, Slawek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for Property

Graphs. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. 2423–2436.
[3] Catriel Beeri, Martin Dowd, Ronald Fagin, and Richard Statman. 1984. On the Structure of Armstrong Relations for Functional Dependencies. *J. ACM* 31, 1 (1984), 30–46. doi:10.1145/2422.322414
[4] Thomas Bläsius, Tobias Friedrich, and Martin Schirneck. 2022. The complexity of dependency detection and discovery in relational databases. *Theor. Comput. Sci.* 900 (2022), 79–96. doi:10.1016/J.TCS.2021.11.020
[5] Tobias Bleifuß, Thorsten Papenbrock, Thomas Bläsius, Martin Schirneck, and Felix Naumann. 2024. Discovering Functional Dependencies through Hitting Set Enumeration. *Proc. ACM Manag. Data* 2, 1 (2024), 43:1–43:24. https://doi.org/10.1145/3639298
[6] B.W. Boehm. 1981. *Software Engineering Economics*. Prentice Hall.
[7] Bernardo Breve, Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. 2024. Decentralized and Incremental Discovery of Relaxed Functional Dependencies Using Bitwise Similarity. *IEEE Trans. Knowl. Data Eng.* 36, 12 (2024), 7380–7398. doi:10.1109/TKDE.2024.3403928
[8] Pieta Brown and Sebastian Link. 2017. Probabilistic Keys. *IEEE Trans. Knowl. Data Eng.* 29, 3 (2017), 670–682. https://doi.org/10.1109/TKDE.2016.2633342
[9] Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan. 2001. Keys for XML. In *Proceedings of the Tenth International World Wide Web Conference, WWW*. 201–210.
[10] Andrea Calì, Diego Calvanese, and Maurizio Lenzerini. 2013. Data Integration under Integrity Constraints. In *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*. 335–352.
[11] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (1970), 377–387.
[12] C. J. Date. 2012. *SQL and Relational Theory - How to Write Accurate SQL Code, Second Edition*. O'Reilly.
[13] János Demetrovics. 1978. On the Number of Candidate Keys. *Inf. Process. Lett.* 7, 6 (1978), 266–269. doi:10.1016/0020-0190(78)90013-3
[14] János Demetrovics and Gy. Gyepesi. 1983. A note on minimal matrix representation of closure operations. *Comb.* 3, 2 (1983), 177–179. doi:10.1007/BF02579291
[15] Rodney G. Downey and Michael R. Fellows. 2013. *Fundamentals of Parameterized Complexity*. Springer. doi:10.1007/978-1-4471-5559-1
[16] Ramez Elmasri and Shamkant B. Navathe. 2000. *Fundamentals of Database Systems, 3rd Edition*. Addison-Wesley-Longman.
[17] A. Enders and H.D. Romback. 2003. *A Handbook of Software and Systems Engineering; Empirical Overservations, Laws and Theories*. Addison-Wesley.
[18] Ronald Fagin. 1982. Horn clauses and database dependencies. *J. ACM* 29, 4 (1982), 952–985. doi:10.1145/322344.322347
[19] Ronald Fagin and Moshe Y. Vardi. 1983. Armstrong Databases for Functional and Inclusion Dependencies. *Inf. Process. Lett.* 16, 1 (1983), 13–19. doi:10.1016/0020-0190(83)90005-4
[20] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering Conditional Functional Dependencies. *IEEE Trans. Knowl. Data Eng.* 23, 5 (2011), 683–698. doi:10.1109/TKDE.2010.154
[21] Peter A. Flach and Iztok Savnik. 1999. Database Dependency Discovery: A Machine Learning Approach. *AI Commun.* 12, 3 (1999), 139–160. http://content.iospress.com/articles/ai-communications/aic182
[22] Georg Gottlob. 2004. Hypergraph Transversals. In *Foundations of Information and Knowledge Systems, Third International Symposium, FoIKS (Lecture Notes in Computer Science, Vol. 2942)*, Dietmar Seipel and Jose Maria Turull Torres (Eds.). Springer, 1–5. doi:10.1007/978-3-540-24627-5_1
[23] Ralf Hartmut Güting. 1994. An Introduction to Spatial Database Systems. *VLDB J.* 3, 4 (1994), 357–399.
[24] Miika Hannula, Xinyi Li, and Sebastian Link. 2023. Controlling entity integrity with key sets. *J. Comput. Syst. Sci.* 136 (2023), 195–219. doi:10.1016/J.JCSS.2023.04.004
[25] Miika Hannula and Jef Wijsen. 2022. A Dichotomy in Consistent Query Answering for Primary Keys and Unary Foreign Keys. In *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. 437–449.
[26] Sven Hartmann, Sebastian Link, and Thu Trinh. 2009. Constraint acquisition for Entity-Relationship models. *Data Knowl. Eng.* 68, 10 (2009), 1128–1155. doi:10.1016/J.DATAK.2009.06.001
[27] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* 42, 2 (1999), 100–111.
[28] Lan Jiang and Felix Naumann. 2020. Holistic primary key and foreign key detection. *J. Intell. Inf. Syst.* 54, 3 (2020), 439–461. doi:10.1007/S10844-019-00562-Z
[29] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. 1988. On Generating All Maximal Independent Sets. *Inf. Process. Lett.* 27, 3 (1988), 119–123.
[30] Vitaliy L. Khizder and Grant E. Weddell. 2003. Reasoning about Uniqueness Constraints in Object Relational Databases. *IEEE Trans. Knowl. Data Eng.* 15, 5 (2003), 1295–1306. doi:10.1109/TKDE.2003.1232279
[31] Warren-Dean Langeveldt and Sebastian Link. 2010. Empirical evidence for the usefulness of Armstrong relations in the acquisition of meaningful functional dependencies. *Inf. Syst.* 35, 3 (2010), 352–374. doi:10.1016/J.IS.2009.11.002
[32] S. Lauesen and O. Vinter. 2001. Preventing Requirement Defects: An Experiment in Process Improvement. *Requir. Eng.* 6, 1 (2001), 37–50.
[33] Georg Lausen. 2007. Relational Databases in RDF: Keys and Foreign Keys. In *Semantic Web, Ontologies and Databases, VLDB Workshop, SWDB-ODBIS 2007, Vienna, Austria, September 24, 2007, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 5005)*, Vassilis Christophides, Martine Collard, and Claudio Gutierrez (Eds.). Springer, 43–56.
[34] Alexander W. Lee, Justin Chan, Michael Fu, Nicolas Kim, Akshay Mehta, Deepti Raghavan, and Ugur Çetintemel. 2025. Semantic Integrity Constraints: Declarative Guardrails for AI-Augmented Data Processing Systems. *Proc. VLDB Endow.* 18, 11 (2025), 4073–4080. https://doi.org/doi:10.14778/3749646.3749677
[35] Claudio L. Lucchesi and Sylvia L. Osborn. 1978. Candidate Keys for Relations. *J. Comput. Syst. Sci.* 17, 2 (1978), 270–279. https://doi.org/10.1016/0022-0000(78)90009-0
[36] Heikki Mannila and Kari-Jouko Räihä. 1986. Design by Example: An Application of Armstrong Relations. *J. Comput. Syst. Sci.* 33, 2 (1986), 126–141.
[37] Fabien De Marchi, Stéphane Lopes, and Jean-Marc Petit. 2009. Unary and n-ary inclusion dependency discovery in relational databases. *J. Intell. Inf. Syst.* 32, 1 (2009), 53–73. doi:10.1007/S10844-007-0048-X
[38] Fabien De Marchi and Jean-Marc Petit. 2007. Semantic sampling of existing databases through informative Armstrong databases. *Inf. Syst.* 32, 3 (2007), 446–457. doi:10.1016/J.IS.2005.12.007
[39] J. Martin. 1989. *Information Engineering*. Prentice Hall.
[40] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection*. Morgan & Claypool Publishers. doi:10.2200/S00262ED1V01Y201003DTM003
[41] Christos H. Papadimitriou. 2007. *Computational complexity*. Academic Internet Publ.
[42] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. *Proc. VLDB Endow.* 8, 12 (2015), 1860–1863. doi:10.14778/2824032.2824086
[43] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 821–833. doi:10.1145/2882903.2915203
[44] Marcel Parciak, Sebastiaan Weytjens, Niel Hens, Frank Neven, Liesbet M. Peeters, and Stijn Vansummeren. 2025. Measuring approximate functional dependencies: a comparative study. *VLDB J.* 34, 4 (2025), 56. doi:10.1007/S00778-025-00931-X
[45] Hemant Saxena, Lukasz Golab, and Ihab F. Ilyas. 2019. Distributed Implementations of Dependency Discovery Algorithms. *Proc. VLDB Endow.* 12, 11 (2019), 1624–1636. doi:10.14778/3342263.3342638
[46] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. 2020. *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company.
[47] Standish Group. 1994. The CHAOS Report. The Standish Group International, available on-line at `http://www.standishgroup.com/sample_research/chaos_1994_1.php`.
[48] Standish Group. 1995. Unfinished Voyages. The Standish Group International, available on-line at `http://www.standishgroup.com/sample_research/unfinsihed_voyages_1.php`.
[49] Yann Strozecki. 2019. Enumeration Complexity. *Bull. EATCS* 129 (2019).
[50] Bernhard Thalheim. 1991. *Dependencies in relational databases*. Teubner-Texte zur Mathematik, Vol. 126. Teubner.
[51] Bernhard Thalheim. 2000. *Entity-relationship modeling - foundations of database technology*. Springer.
[52] Ziheng Wei, Uwe Leck, and Sebastian Link. 2019. Discovery and Ranking of Embedded Uniqueness Constraints. *Proc. VLDB Endow.* 12, 13 (2019), 2339–2352. doi:10.14778/3358701.3358703
[53] Ziheng Wei and Sebastian Link. 2018. DataProf: Semantic Profiling for Iterative Data Cleansing and Business Rule Acquisition. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 1793–1796. doi:10.1145/3183713.3193544
[54] Ziheng Wei and Sebastian Link. 2019. Discovery and Ranking of Functional Dependencies. In *35th IEEE International Conference on Data Engineering, ICDE*. IEEE, 1526–1537. doi:10.1109/ICDE.2019.00137
[55] Jef Wijsen. 1993. A Theory of Keys for Temporal Databases. In *Neuvièmes Journées Bases de Données Avancées*. 35–54.
[56] R.E. Zultner. 1992. The Deming Way: Total Quality Management for Software. In *Proceedings of Total Quality Management for Software*. 134–145.